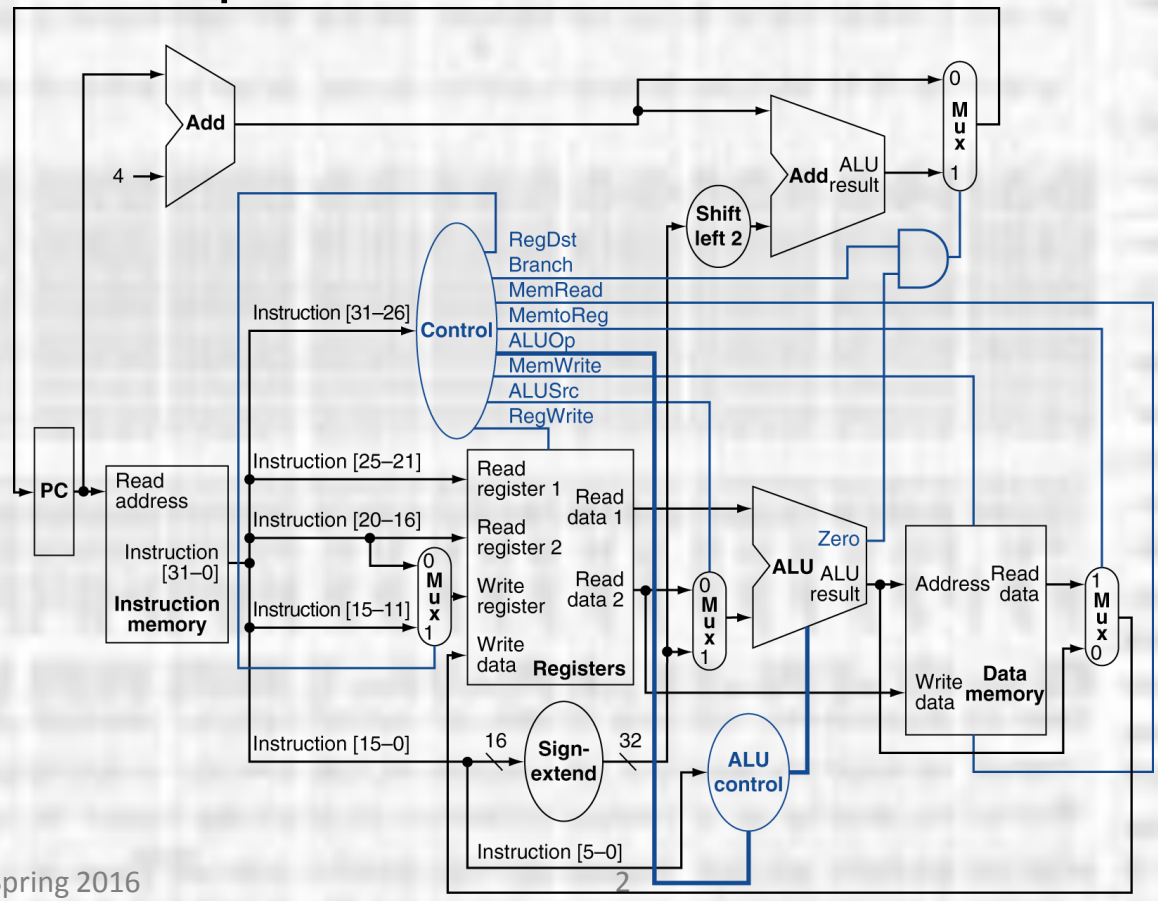
ELE 455/555
Computer System Engineering

Section 2 – The Processor

Class 4 – Pipelining with Hazards

Pipelining
Overview

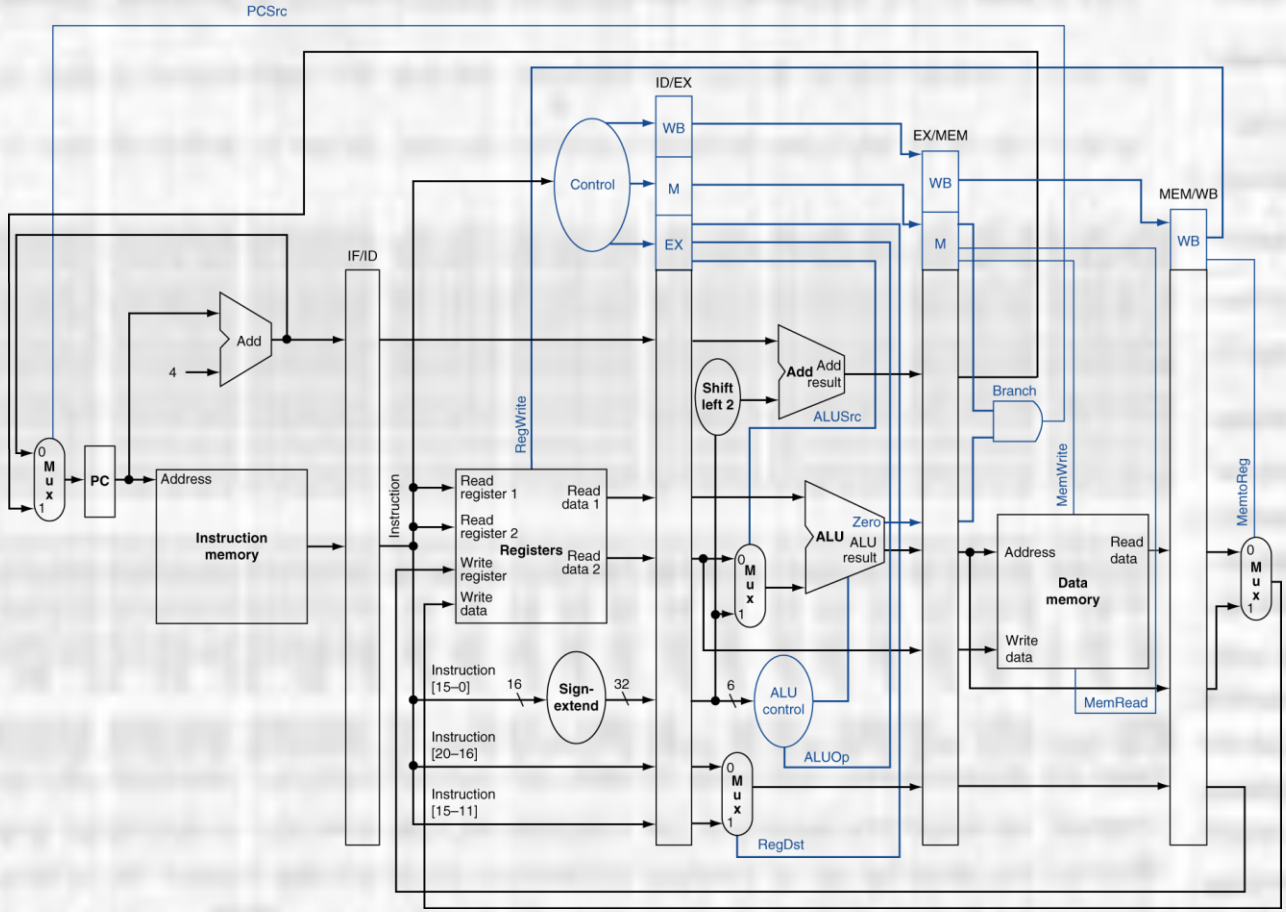
- Simple Datapath



Pipelining

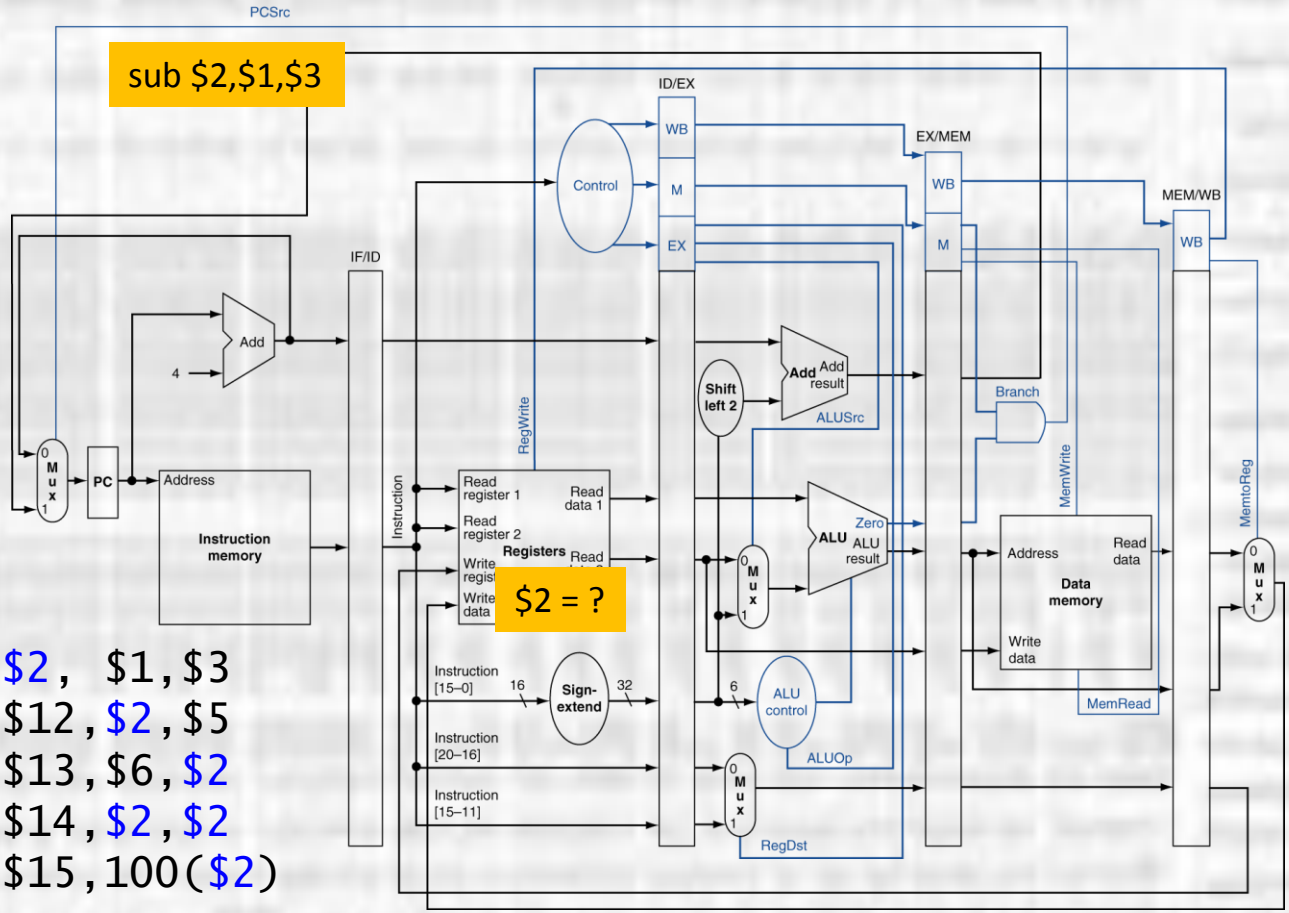
Overview

- Pipeline Control



Pipelining
Data Hazards

- Data Hazards




```
sub $2, $1,$3
and $12,$2,$5
or  $13,$6,$2
add $14,$2,$2
sw  $15,100($2)
```

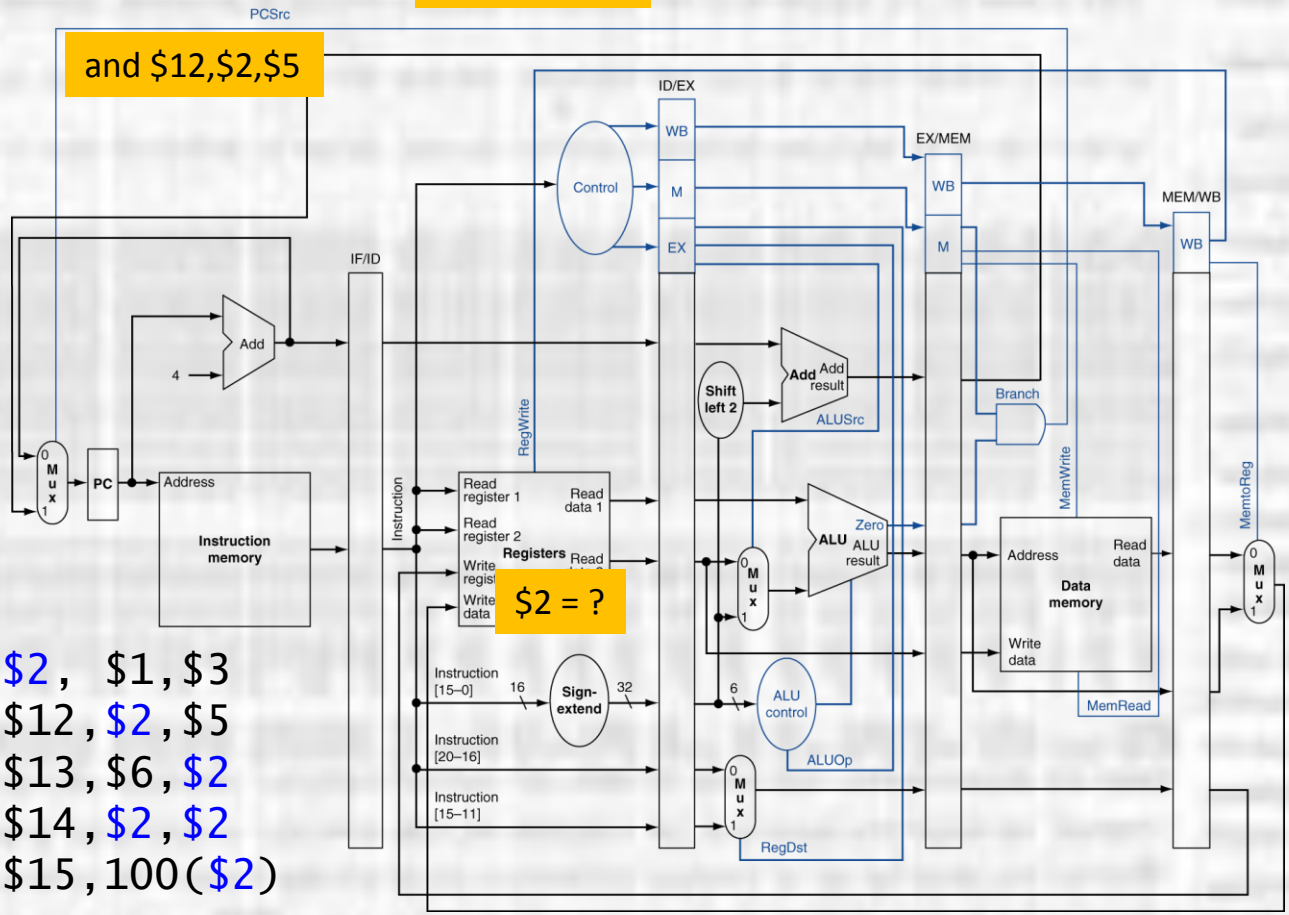
Pipelining

Data Hazards

- Data Hazards

sub $2,$1,$3

and $12,$2,$5




```
sub $2, $1,$3
and $12,$2,$5
or  $13,$6,$2
add $14,$2,$2
sw  $15,100($2)
```

Pipelining

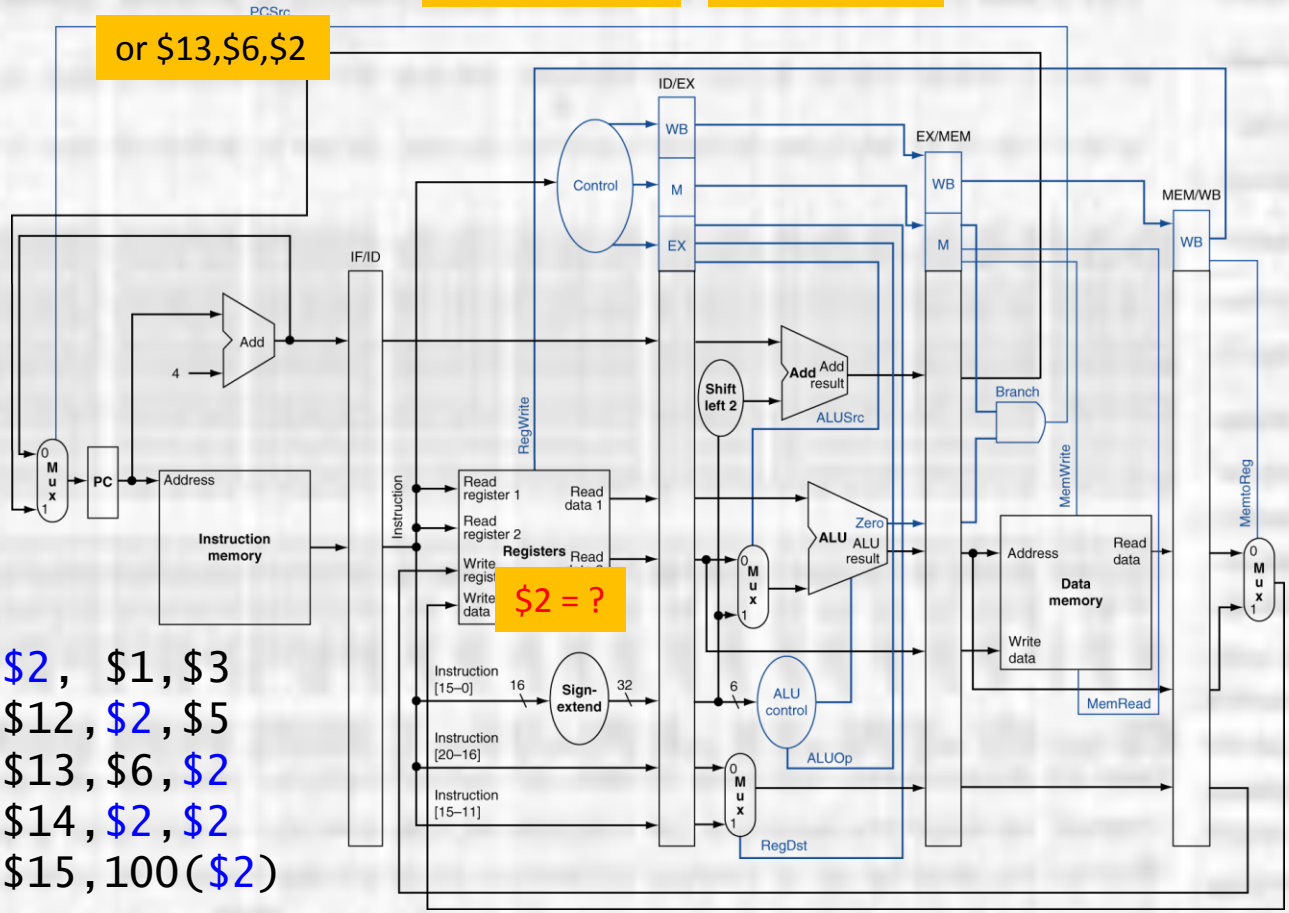
Data Hazards

- Data Hazards

and $12,$2,$5

sub $2,$1,$3

or $13,$6,$2

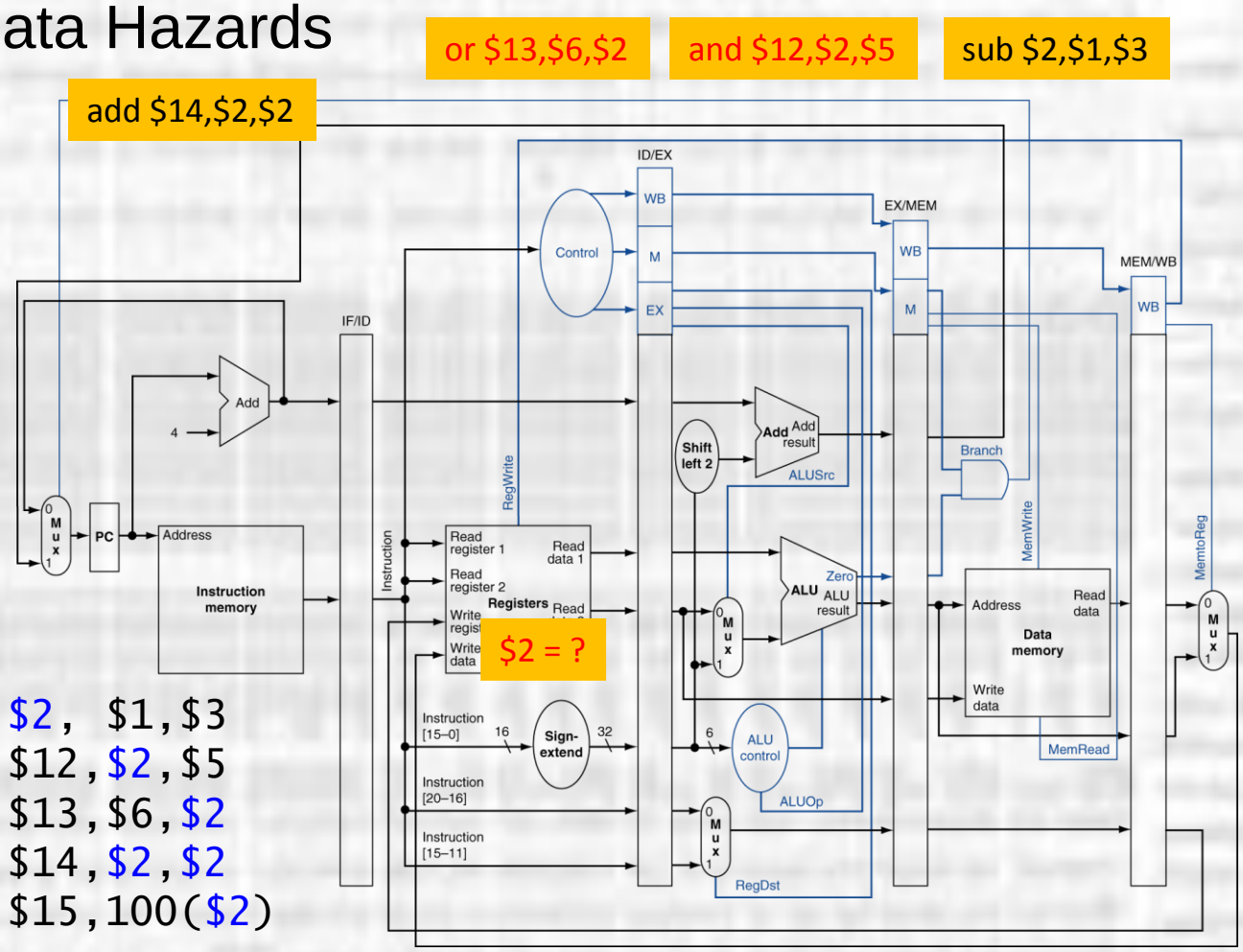



```
sub $2, $1,$3
and $12,$2,$5
or  $13,$6,$2
add $14,$2,$2
sw  $15,100($2)
```

Pipelining

Data Hazards

- Data Hazards




```
sub $2, $1,$3
and $12,$2,$5
or  $13,$6,$2
add $14,$2,$2
sw  $15,100($2)
```

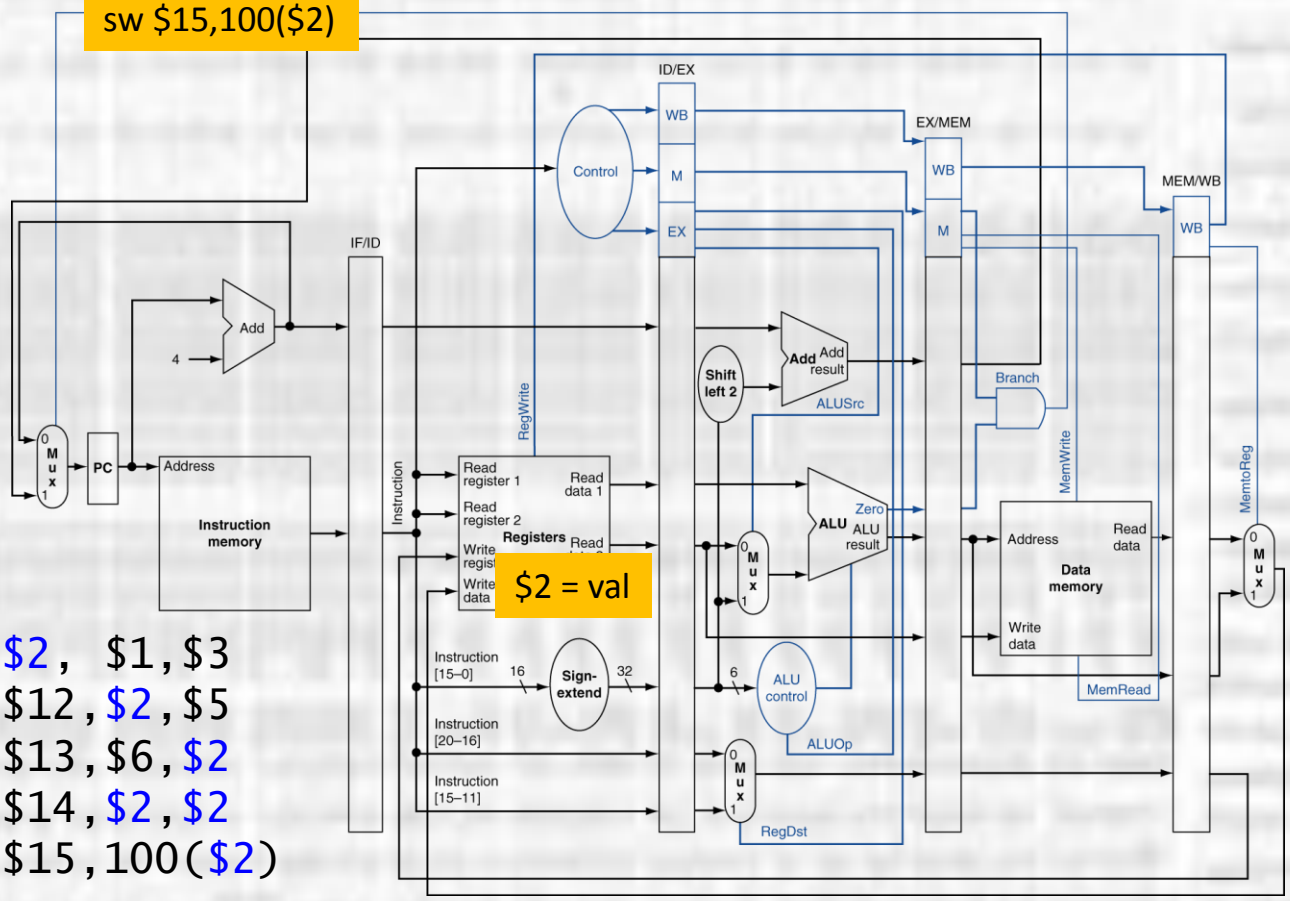
Pipelining

Data Hazards

- Data Hazards

add $14,$2,$2 | or $13,$6,$2 | and $12,$2,$5 | sub $2,$1,$3

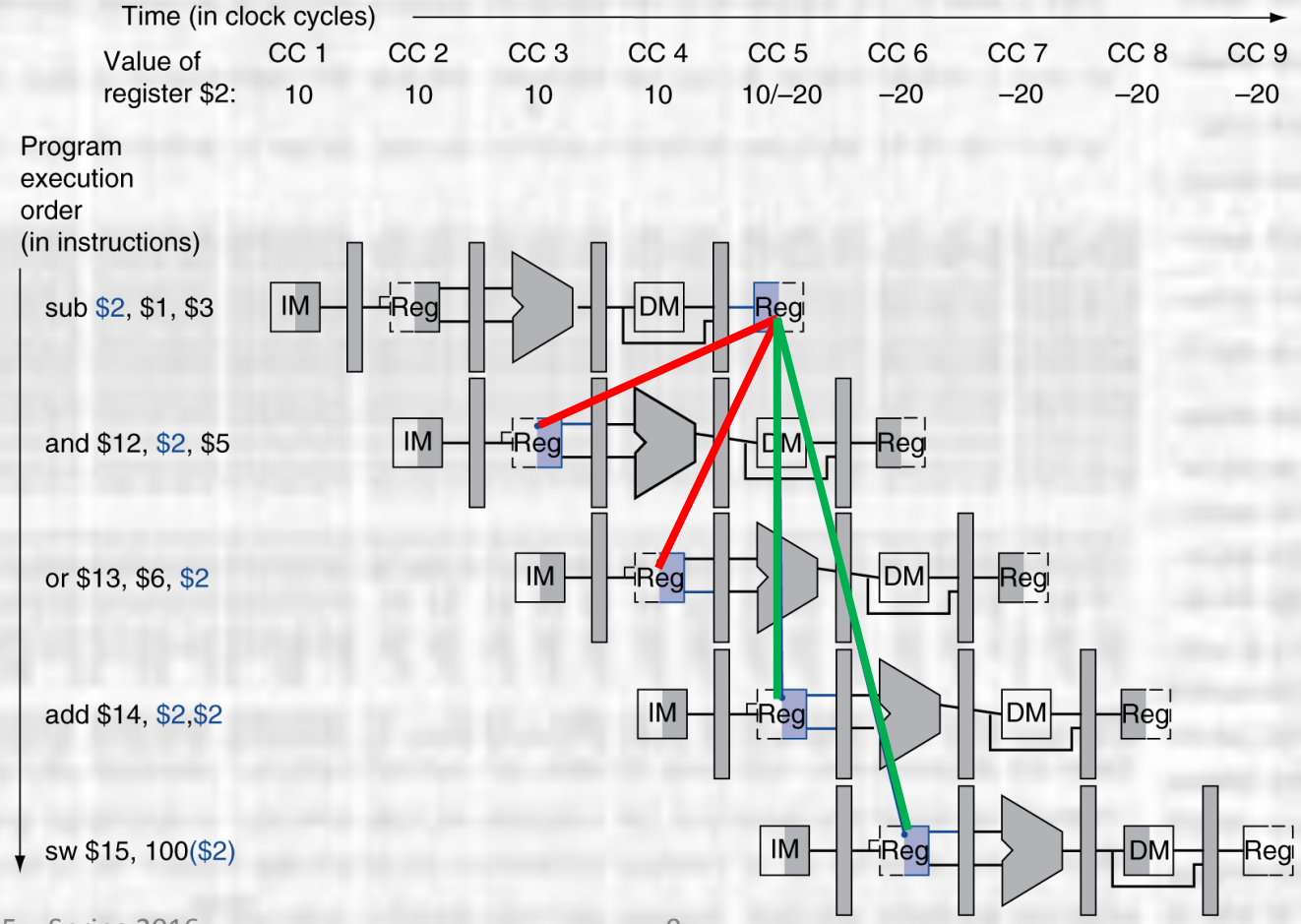
sw $15,100($2)




```
sub $2, $1,$3
and $12,$2,$5
or  $13,$6,$2
add $14,$2,$2
sw  $15,100($2)
```

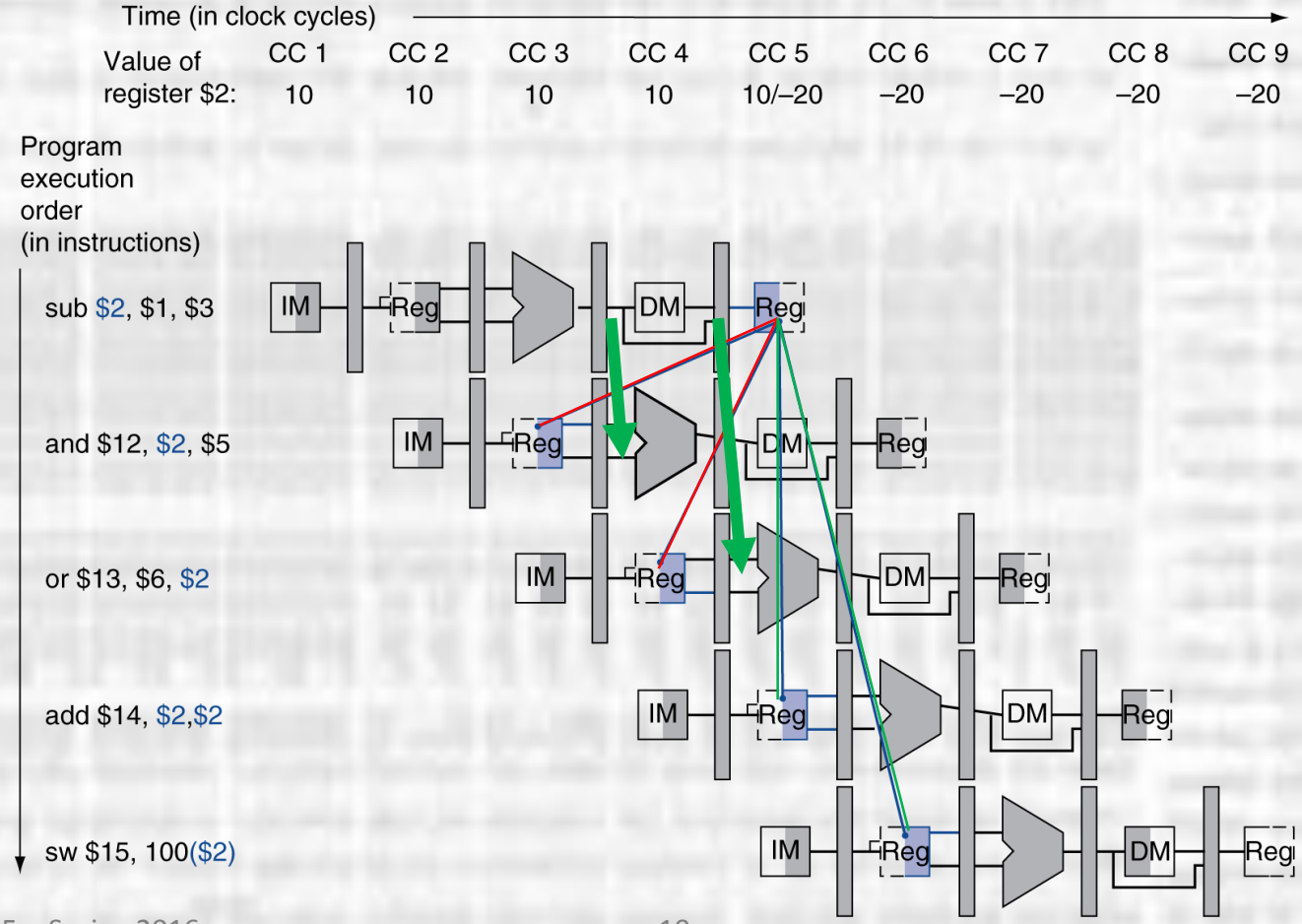
Pipelining
Data Hazards

- Data Hazards



Pipelining
Data Hazards

- Forwarding



Pipelining
Data Hazards

- Detecting the need for Forwarding

  - Terminology:
    - ID/EX.RegisterRs = register # whose value is found in register ID/EX from read data 1 in the register file
    - ID/EX.RegisterRt = register # whose value is found in register ID/EX from read data 2 in the register file

  Eg. sub $2, $1, $3 // in execute stage
  ID/EX.RegisterRs = $1
  ID/EX.RegisterRt = $3

    - EX/MEM.RegisterRd= destination register # whose value is found in register EX/MEM
    - MEM/WB.RegisterRd= destination register # whose value is found in register MEM/WB

Pipelining
Data Hazards

- Detecting the need for Forwarding
  - Conditions:
    - 1a) EX/MEM.RegisterRd = ID/EX.RegisterRs
      - EX/MEM currently holds a value needed by an instruction about to enter EX
    - 1b)) EX/MEM.RegisterRd = ID/EX.RegisterRt
      - EX/MEM currently holds a value needed by an instruction about to enter EX
    - 2a) MEM/WB.RegisterRd = ID/EX.RegisterRs
      - MEM/WB currently holds a value needed by an instruction about to enter EX
    - 2b) MEM/WB.RegisterRd = ID/EX.RegisterRt
      - MEM/WB currently holds a value needed by an instruction about to enter EX

Pipelining

Data Hazards

- Detecting the need for Forwarding

```
sub $2, $1,$3
and $12,$2,$5
or  $13,$6,$2
add $14,$2,$2
sw  $15,100($2)
```

- sub-and is a type 1a hazard
  - and Rs ($2) needs result of sub (Rd in EX/MEM)
- sub-or is a type 2b hazard
  - or Rt ($2) needs result from sub (Rd in MEM/WB)
- sub-add is not a hazard due to write/read policy

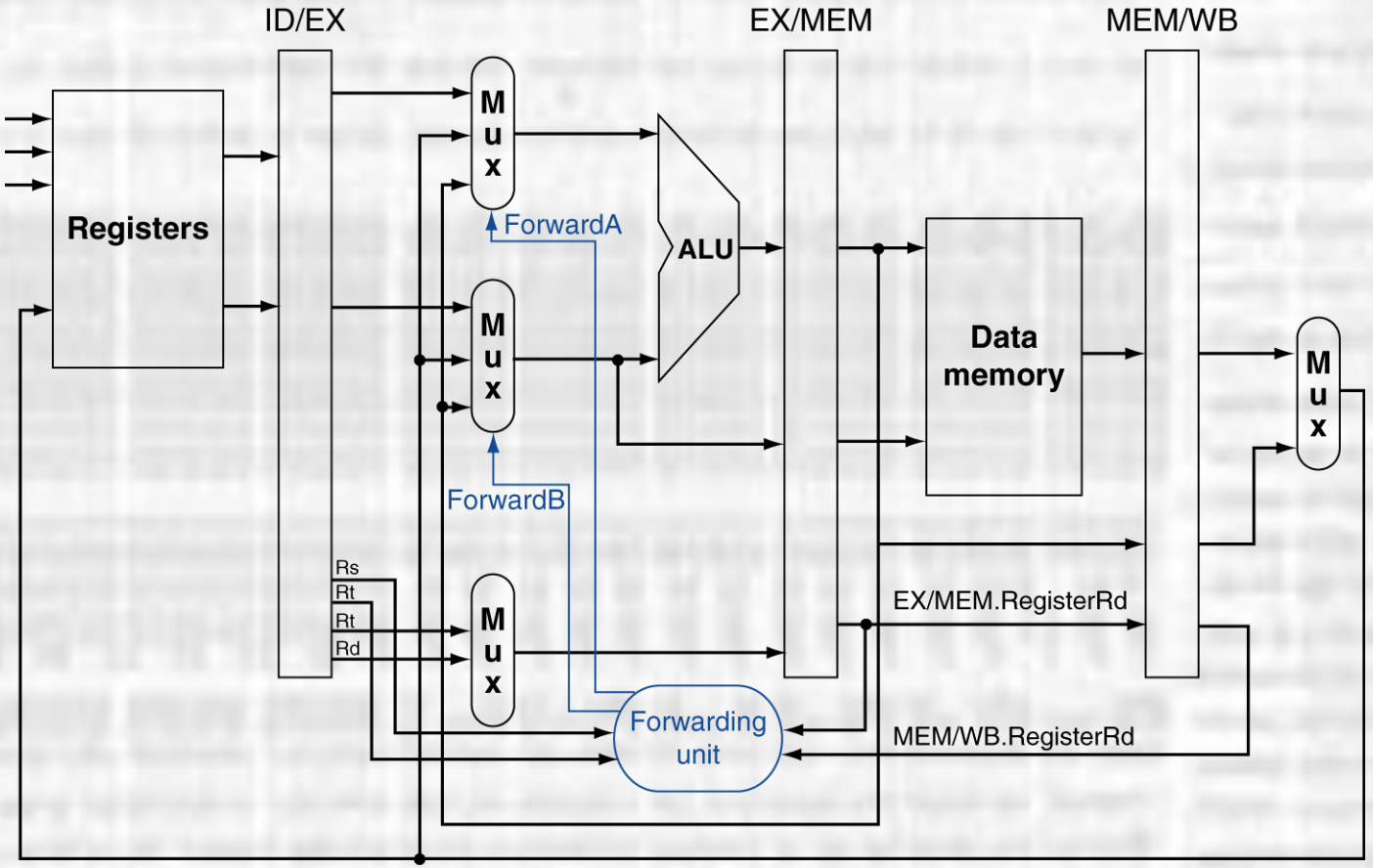
Pipelining
Data Hazards

- Implementing Forwarding
  - Need Rd in EX/MEM and MEM/WB registers
    - We are already doing this to support proper WB
  - Need to multiplex the ALU output from EX/MEM and the WB output from MEM/WB into the ALU inputs
  - Need control to manage the decision process
    - Inputs: Rs, Rt, Rd from EX/MEM, Rd from MEM/WB
      aka: ID/EX.RegisterRs, ID/EX.RegisterRt,
      EX/MEM.RegisterRd, MEM/WB.RegisterRd)
    - Outputs: ALU input mux control A, ALU input mux control B

Pipelining

Data Hazards

- Implementing Forwarding



b. With forwarding

Pipelining
Data Hazards

- Implementing Forwarding
  - 2 additional conditions for forwarding
  - Only forward when a register write is part of the instruction
  - Only forward when the register write is not to register $0

Pipelining
Data Hazards

- Implementing Forwarding
  - EX hazard
    - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
      and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
      ForwardA = 10
    - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
      and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
      ForwardB = 10
  - MEM hazard
    - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
      and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
      ForwardA = 01
    - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
      and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
      ForwardB = 01

Pipelining
Data Hazards

- Implementing Forwarding
  - 1 more condition
  - Double Data Hazard

```
add $1,$1,$2
add $1,$1,$3
add $1,$1,$4
```

  - Both hazards occur
  - Want to use the most recent
    - Revise MEM hazard condition
    - Only forward if EX hazard condition isn't true

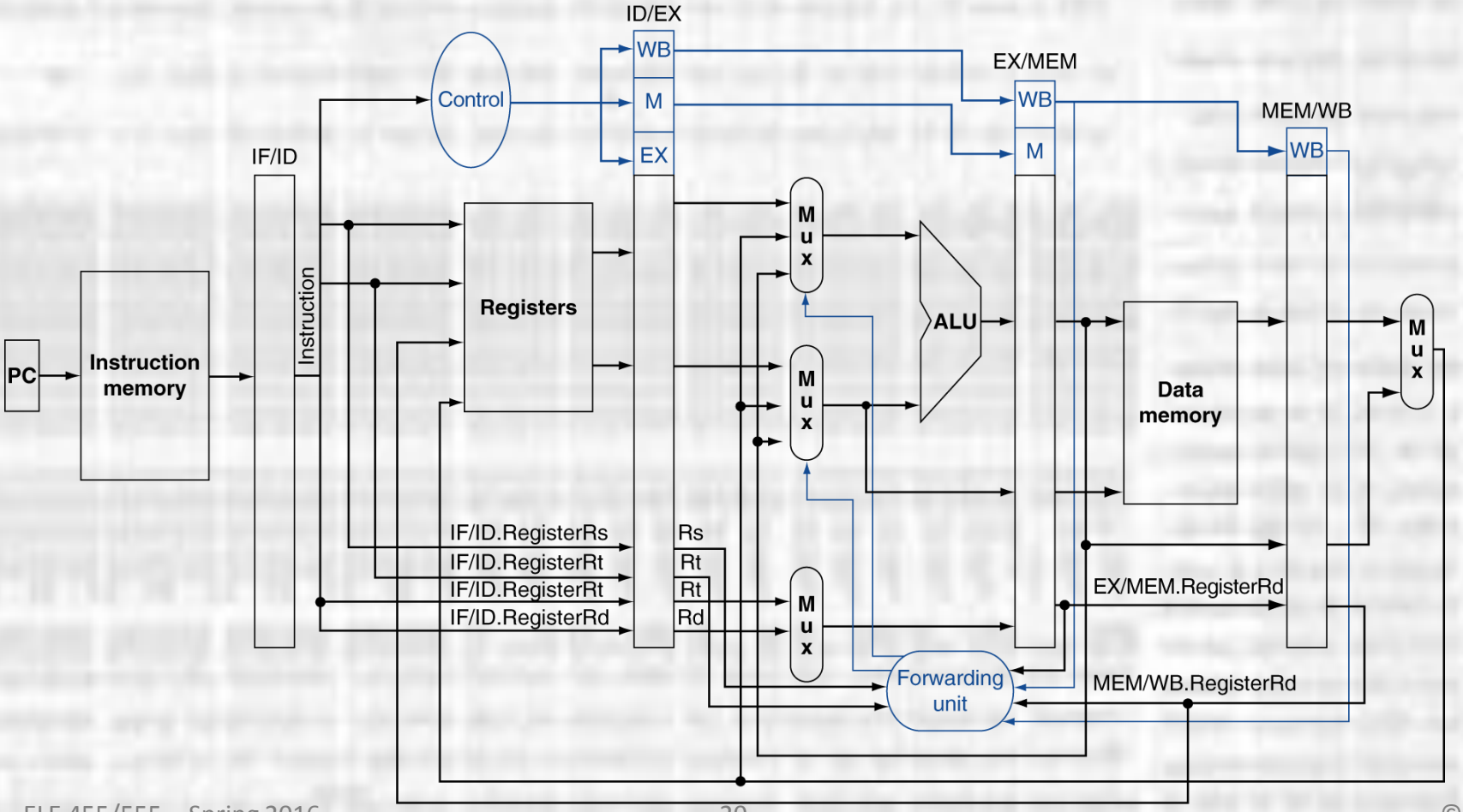
Pipelining
Data Hazards

- Implementing Forwarding
  - EX hazard
    - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
      and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
      ForwardA = 10
    - if (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
      and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
      ForwardB = 10
  - MEM hazard
    - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
      and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
      and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
      and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
      ForwardA = 01
    - if (MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
      and not (EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
      and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
      and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
      ForwardB = 01

Pipelining

Data Hazards

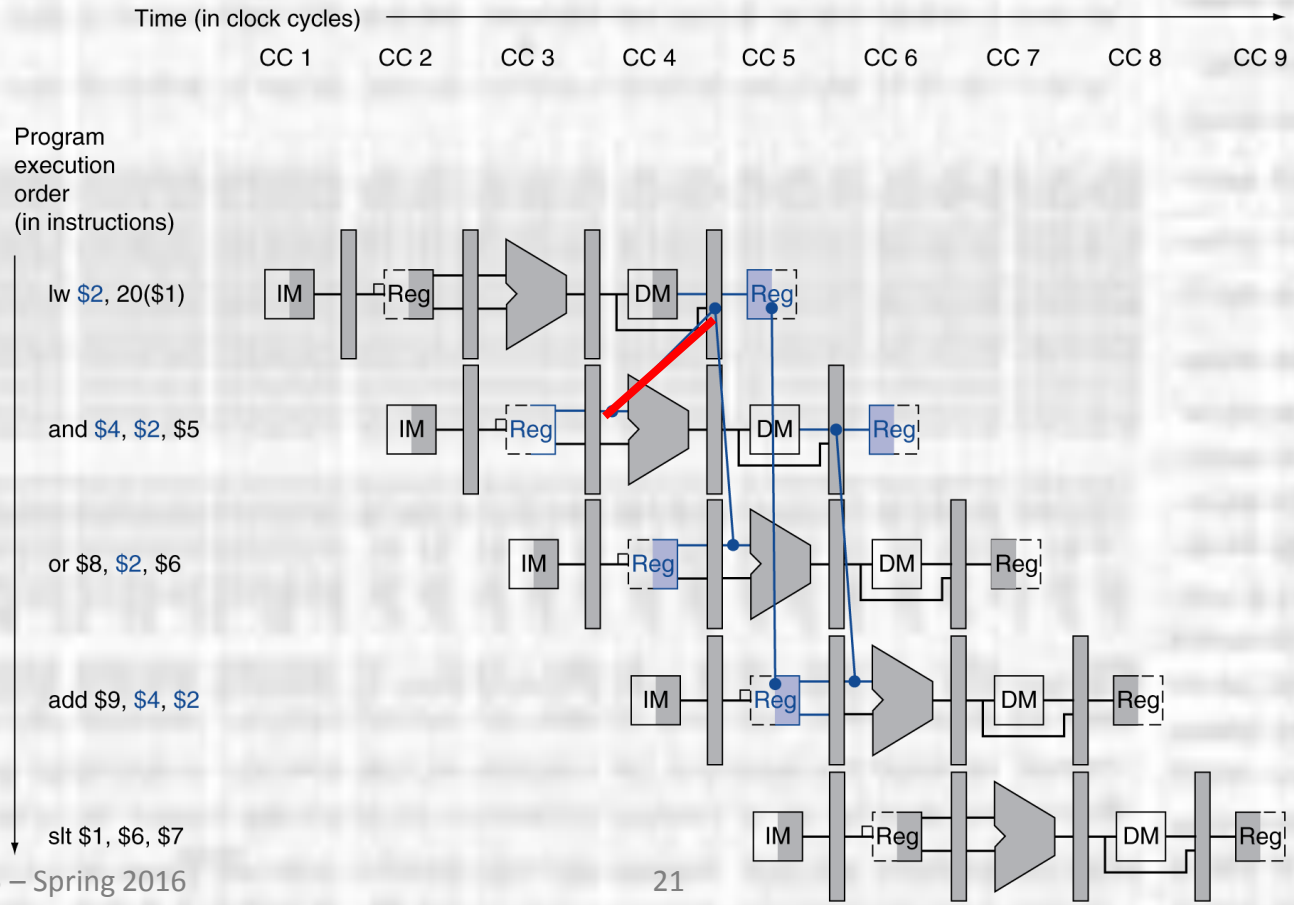
- Implementing Forwarding



Pipelining

Stall

- Stalling the pipeline and inserting a bubble
  - Consider a load-use hazard



Pipelining

Stall

- Stalling the pipeline and inserting a bubble
  - Load-use hazard
  - Cannot be resolved with forwarding → stall

  - Check for a memory read value in an instruction in the execute stage
  - Do this in the ID stage

  - Load-use hazard:

    ID/EX.MemRead and
    ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
    (ID/EX.RegisterRt = IF/ID.RegisterRt))

  - When true – stall the pipeline

Pipelining

Stall

- Stalling the pipeline and inserting a bubble
  - Force the control values (for EX, MEM, and WB) to “0” in the ID/EX stage
  - “0” for the control signals causes nothing to happen
  - Prevent the PC from updating
  - Prevent the IF/ID register from updating

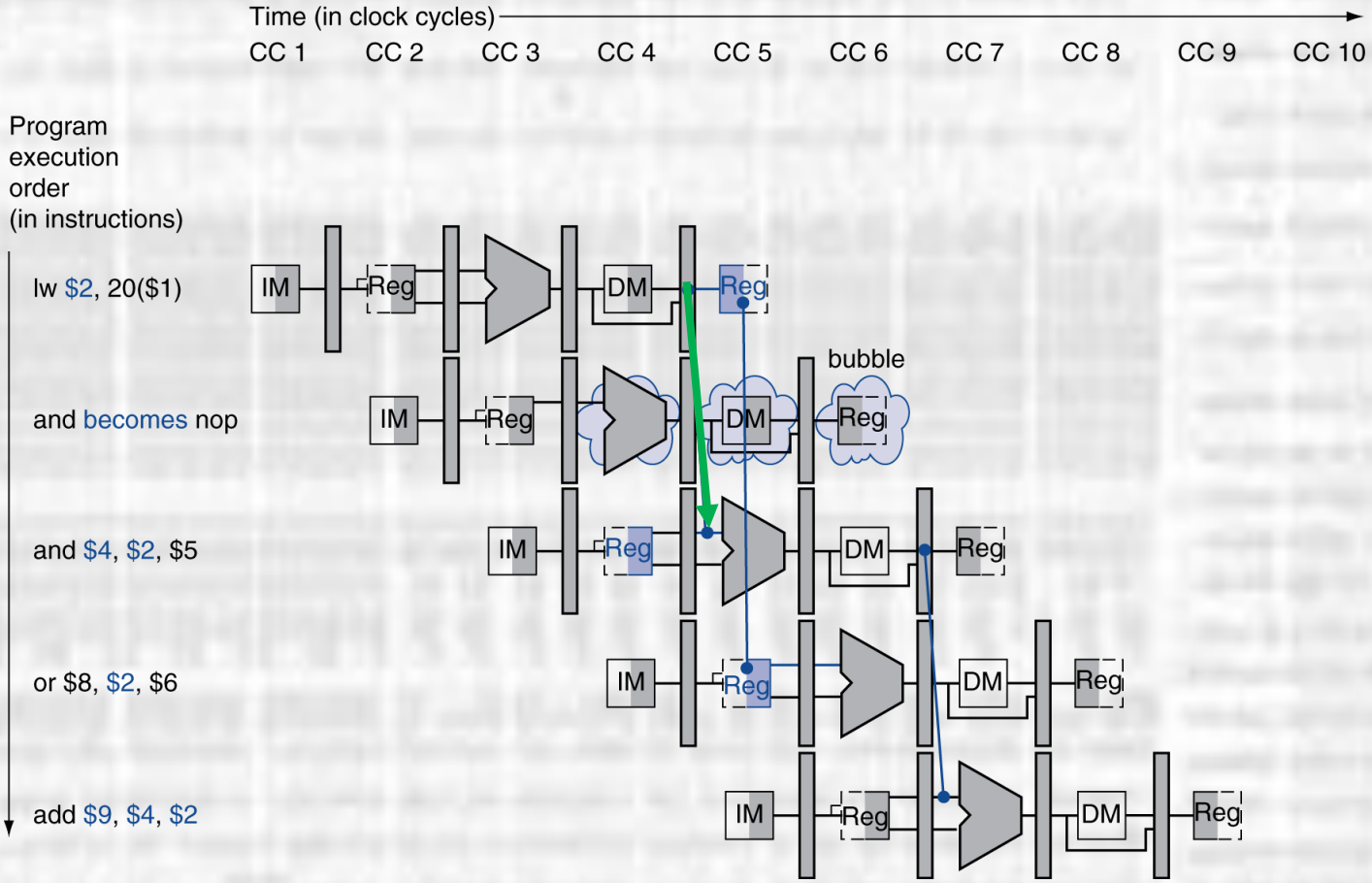
→ On next clock – will reload the same value into IF/ID

At the same time the LW instruction progresses to the MEM stage and the value is available to be forwarded

Pipelining

Stall

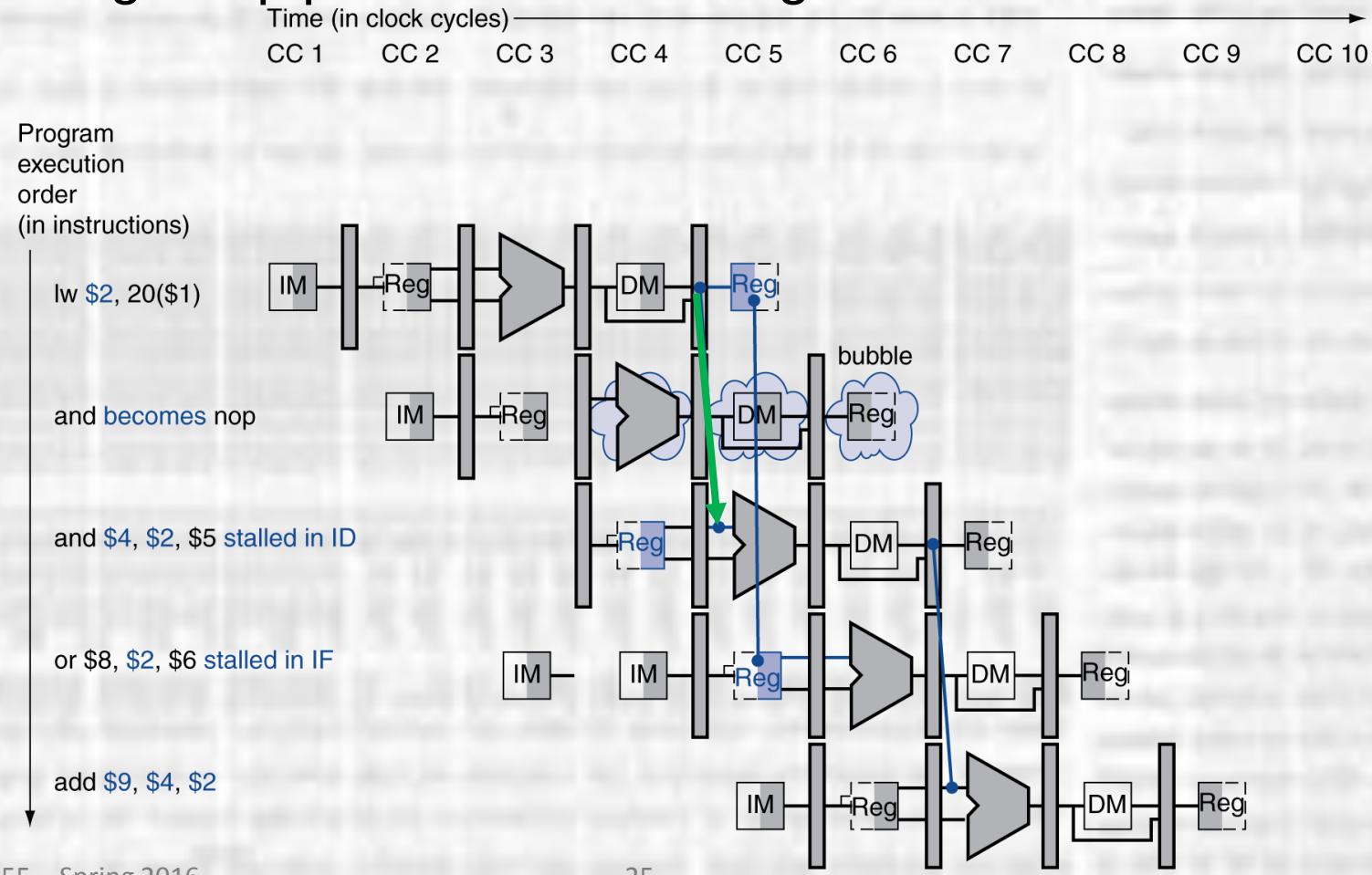
- Stalling the pipeline and inserting a bubble



Pipelining

Stall

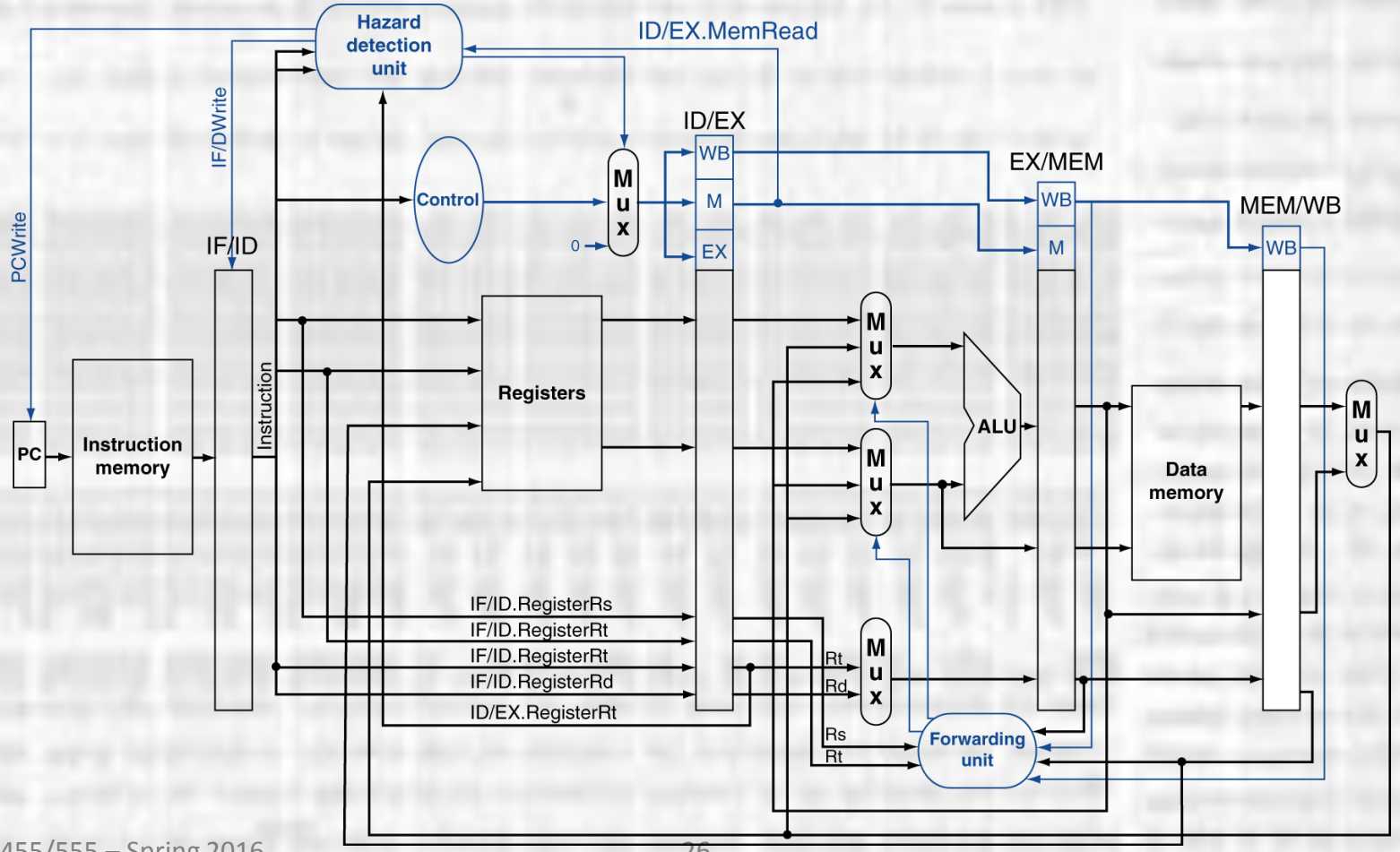
- Stalling the pipeline and inserting a bubble



Pipelining

Stall

- Stalling the pipeline and inserting a bubble



Pipelining
Control Hazards

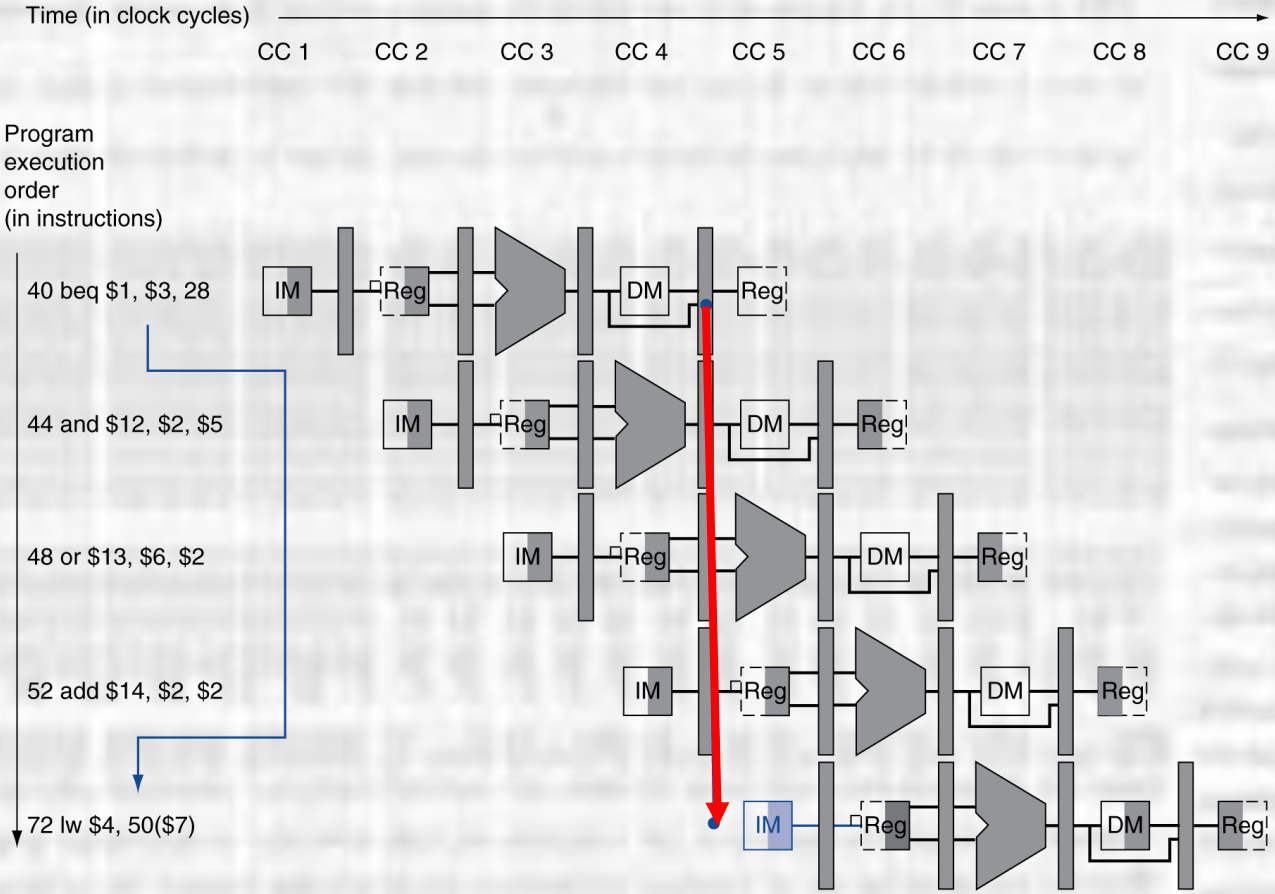
- Branch Hazard
  - Consider the following code snippit

```
        beq     $1, $3, skip
        and     $13, $6, $2
        add     $14, $2, $2
skip    lw      $4, 50($7)
```

  - The branch decision is known after the calculation of the branch address and the comparison (subtract and check for zero), and is available in the MEM stage
  - If the branch is ignored – we will have the and, add and lw instructions in the pipeline – all is well
  - If the branch is taken we will have the and, add and lw instructions in the pipeline – but we do not want them to execute

Pipelining
Control Hazards

- Branch Hazard

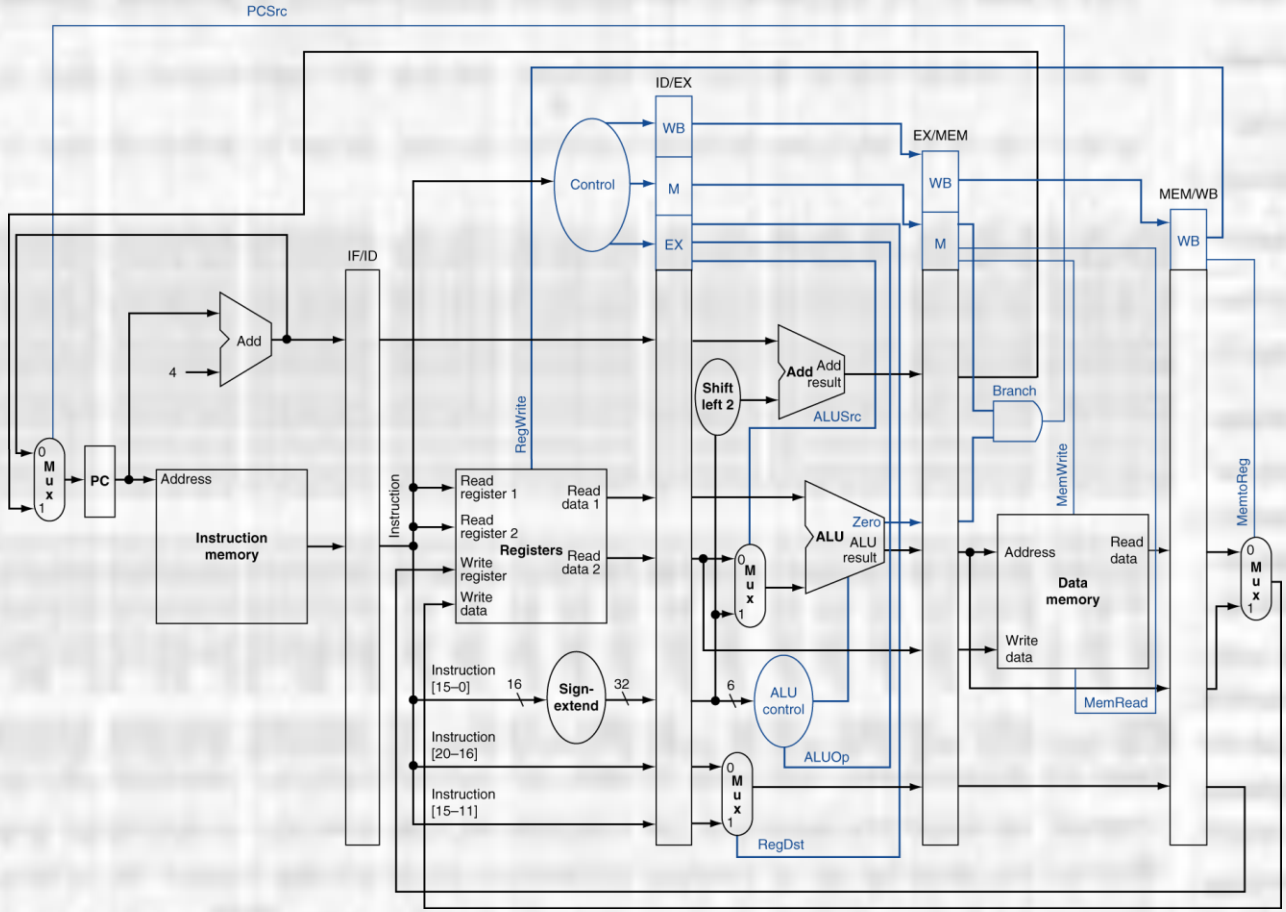


Pipelining
Control Hazards

- Branch Hazard
  - In our current implementation
    - We assume branches are not taken
    - We would need to flush the pipeline for taken branches
      - Branch decision is available in the MEM stage
      - Assuming the branch target is not already in the pipeline
      - → inserting 3 bubbles into the pipeline
  - ? – How far can we move the decision forward to reduce the impact of taken branches

Pipelining
Overview

- Pipeline Control



Pipelining
Control Hazards

- Branch Hazard
  - Most branches are simple comparisons
    - equal → all bits the same
    - negative/positive → look at msb
    - We can move most of the branch prediction logic forward to the ID stage
      - We have register values (some may be forwarded!)
      - Need to modify the forwarding logic to account for branches
    - We can move the branch address calculation forward to the ID stage
    - Still have a single cycle stall – IF of the next instruction is occurring in parallel with ID detection of the taken branch

Pipelining
Control Hazards

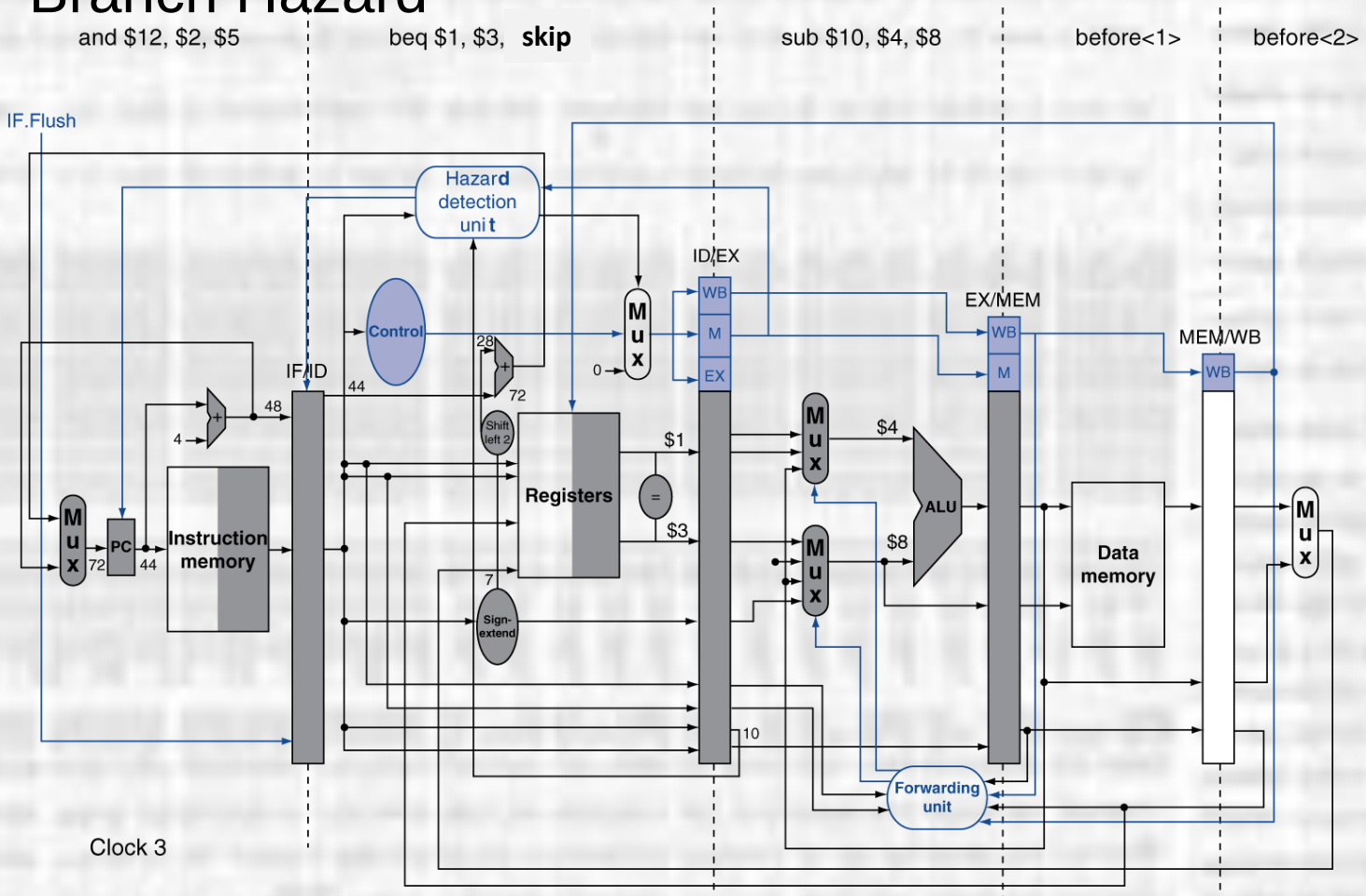
- Branch Hazard
  - Consider this code snippit

```
        sub     $10, $4, $8
        beq     $1, $3, skip
        and     $12, $2, $5
        or      $13, $2, $6
        add     $14, $4, $2
        …
skip    lw      $4, 50($7)
```

Pipelining

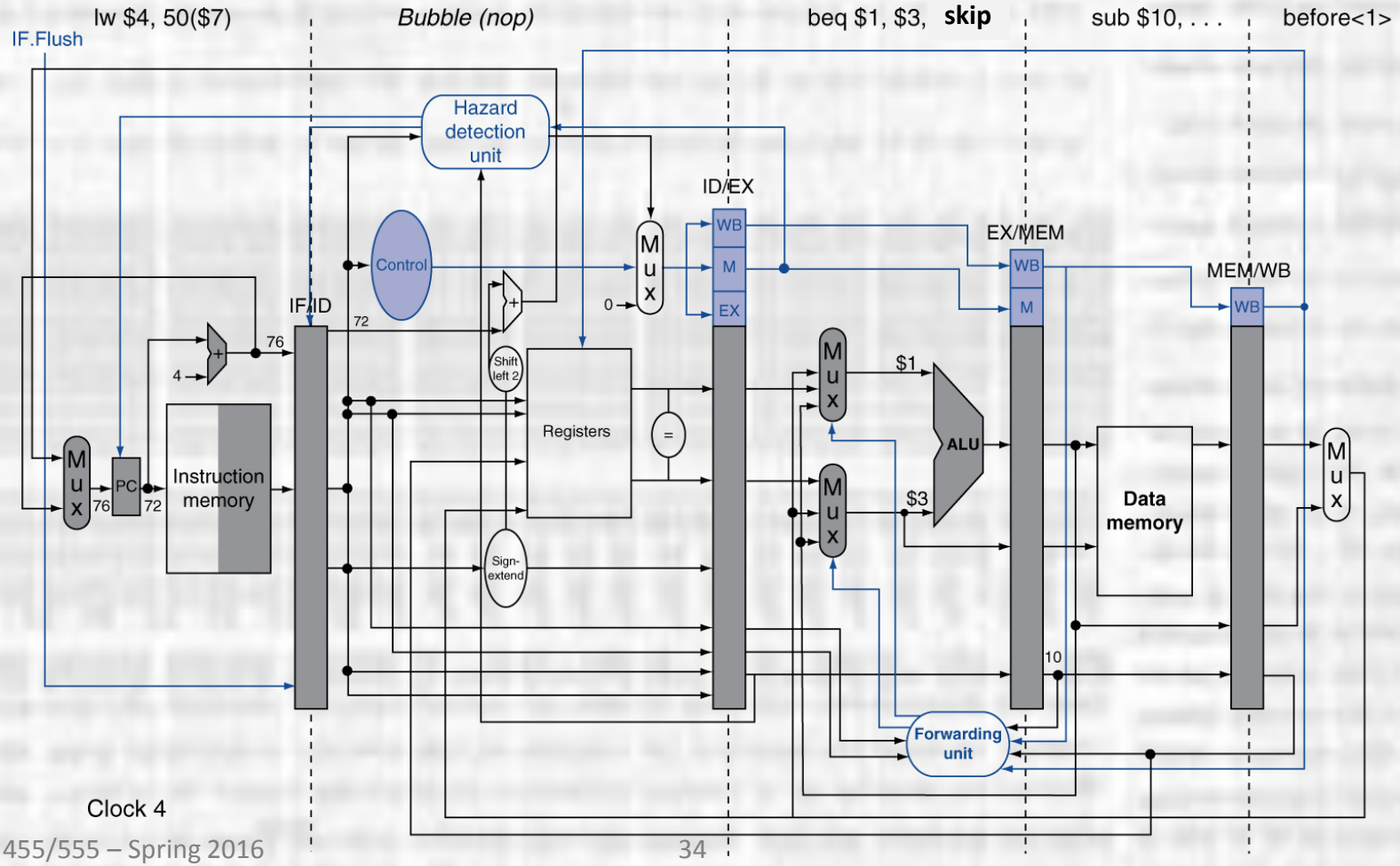
Control Hazards

- Branch Hazard



Pipelining
Control Hazards

- Branch Hazard

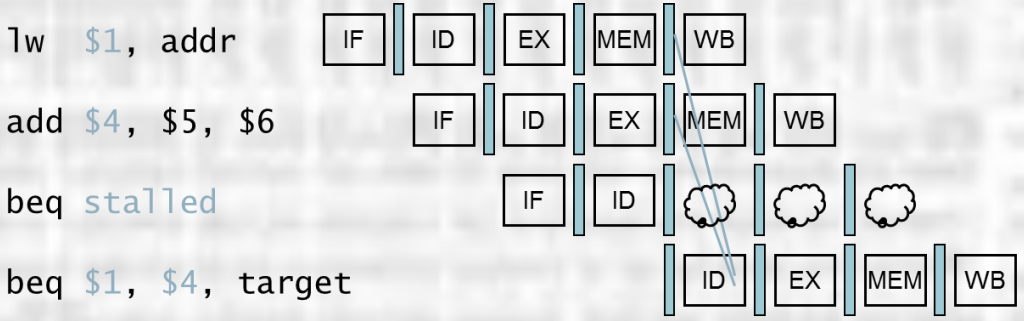


Pipelining
Control Hazards

- Branch Hazard
  - This approach reduces the impact of branch hazards
  - There may still be data hazards that cannot be avoided
    - Branch dependent on ALU output from previous instruction

```
add     $4, $5, $6
beq     $1, $4, skip
```

during the beq ID stage, the add is in EX stage and $4 is not available
→ 1 stall cycle



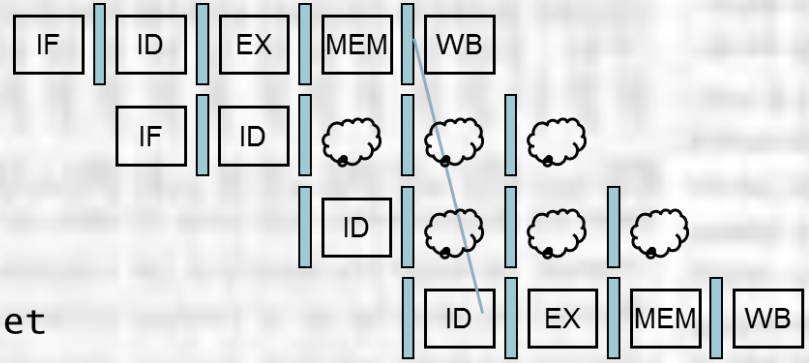
Pipelining
Control Hazards

- Branch Hazard
  - This approach reduces the impact of branch hazards
  - There may still be data hazards that cannot be avoided
    - Branch dependent on previous lw instruction

```
lw      $1, 50($7)
beq     $1, $4, skip
```

the lw result will not be available until the MEM cycle is complete
→ 2 stall cycles

```
lw  $1, addr
beq stalled
beq stalled
beq $1, $0, target
```



Pipelining

Control Hazards

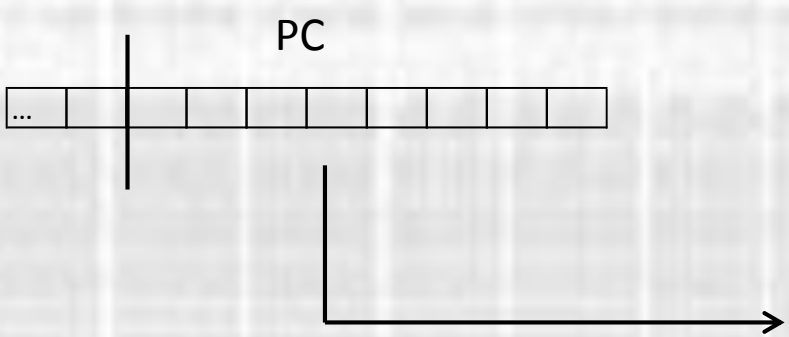
- Branch Prediction
  - For deeper pipelines – the cost of missing a branch decision can be significant – many clock cycles lost
  - Leads to dynamic branch prediction
    - Keep track if branch was previously taken or not
    - Load either the next instruction or the target instruction based on prediction value

Pipelining
Control Hazards

- Dynamic Branch Prediction – 1 bit
  - Use a small branch prediction buffer
    - n words deep
    - 1 bit of prediction value (1 bit word)
  - n is derived from the PC value
    - last 8 bits of PC → 256 words deep
  - The PC value references one of n predictions values
    - Assuming a branch instruction
    - Take the branch if the prediction value is set to 1
    - Don't take the branch if the prediction value is set to 0
  - If the prediction was wrong – invert the prediction value

Pipelining
Control Hazards

- Dynamic Branch Prediction – 1 bit



Branch Table

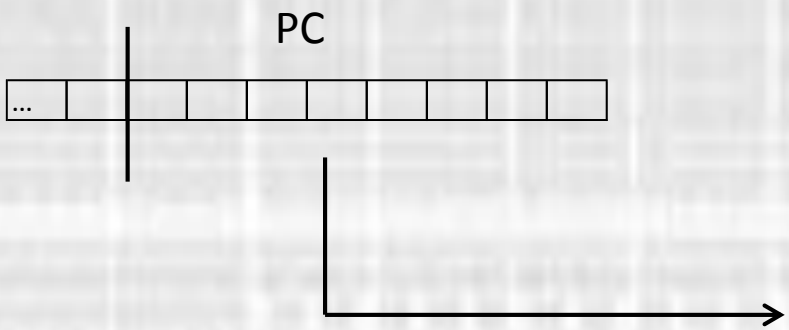
ADDR	B
0x00000000	0
0x00000001	0
0x00000010	1
...	
0x11111110	1
0x11111111	1

Pipelining
Control Hazards

- Dynamic Branch Prediction – 1 bit
  - Issues
    - Multiple PC values point to the same branch table location
      - over write each other
        → wrong guesses
    - Each incorrect guess can lead to 2 wrong guesses
      - eg. Assume mostly loop back – bit set to 1
        when you do not loop back – you stall and set bit to 0
        next cycle you want to loop back but bit is 0 – stall and set bit to 1
      - 2 stalls

Pipelining
Control Hazards

- Dynamic Branch Prediction – 2 bit
  - Use 2 bits to make prediction decisions
    - Only change the prediction on 2 successive mispredictions
    - Resolves the 2 stall issue of 1 bit prediction

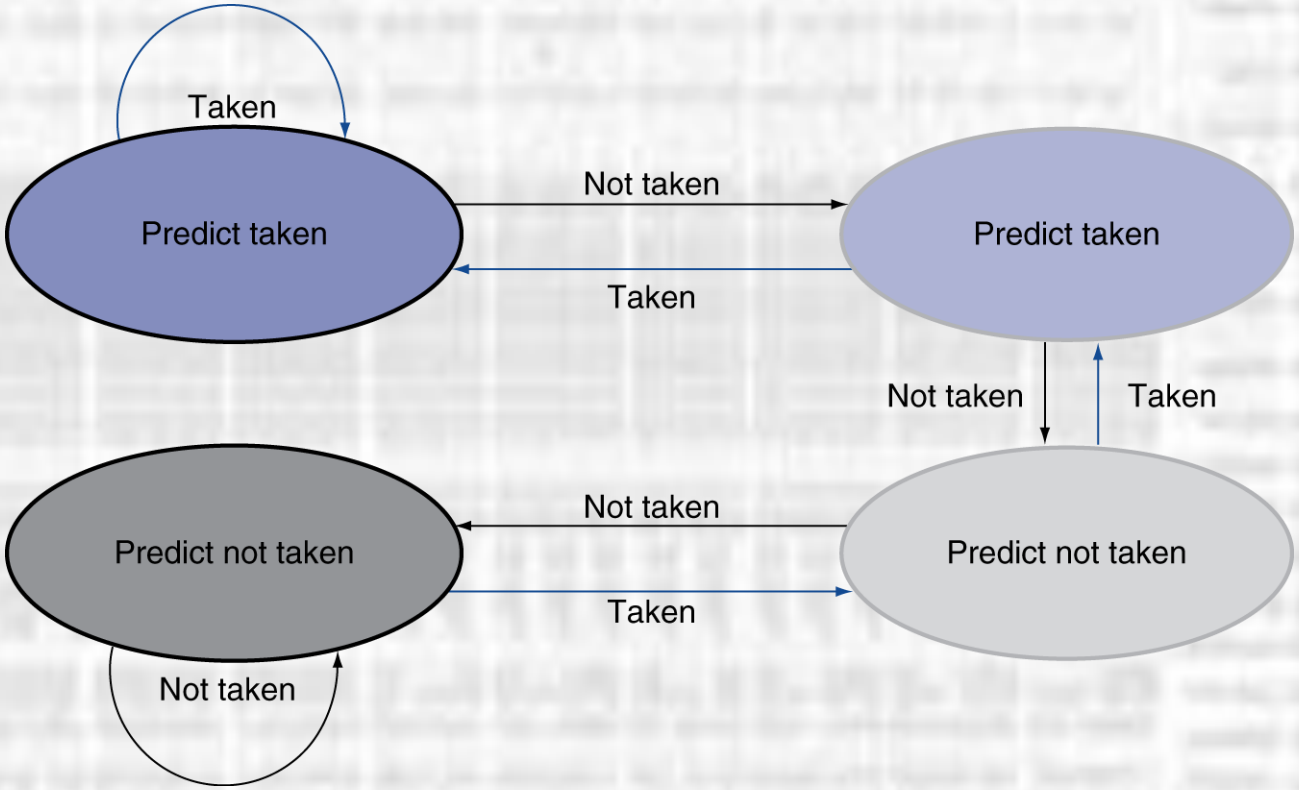


Branch Table

ADDR	B1	B0
0x00000000	0	0
0x00000001	0	0
0x00000010	1	0
...		
0x11111110	0	1
0x11111111	1	1

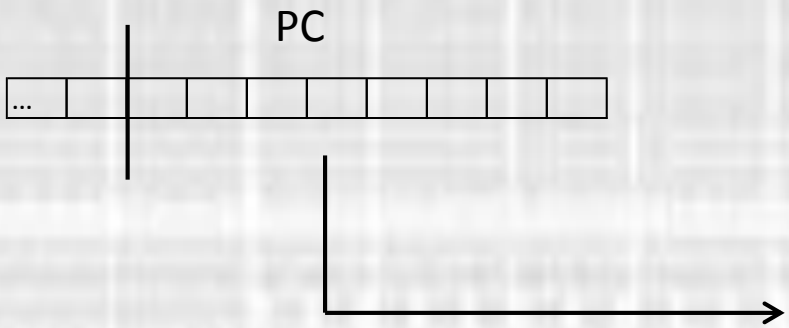
Pipelining
Control Hazards

- Dynamic Branch Prediction – 2 bit



Pipelining
Control Hazards

- Dynamic Branch Prediction – n bit
  - Use n bits to make prediction decisions
    - Use a saturated up/down counter for each table entry
    - Only change prediction when MSB toggles



Branch Table

ADDR	B7	B6	B5	B4	B3	B2	B1	B0
0x00000000	0	0	0	1	0	0	1	0
0x00000001	0	1	1	0	0	0	0	0
0x00000010	1	0	1	0	1	0	1	0
...								
0x11111110	0	1	0	1	0	1	0	1
0x11111111	1	1	0	1	0	1	1	0